

CLA 使用指南

EasyDsp 开发套件

V1.00 Date: 2010/1/13 产品用户手册

类别	内容
关键词	CCS v4.x，CLA 调试
摘　要	本文致力于详细介绍 CCS v4.x 开发环境下 CLA 的调试方法。

广州致远电子有限公司



修订历史

版本	日期	原因
V1.00	2010/1/13	创建文档

销售与服务网络（一）

广州周立功单片机发展有限公司

地址：广州市天河北路 689 号光大银行大厦 12 楼 F4
邮编：510630
电话：(020)38730916　38730917　38730972　38730976　38730977
传真：(020)38730925
网址：www.zlgmcu.com

广州专卖店

地址：广州市天河区新赛格电子城 203-204 室
电话：(020)87578634　87569917
传真：(020)87578842

南京周立功

地址：南京市珠江路 280 号珠江大厦 2006 室
电话：(025)83613221　83613271　83603500
传真：(025)83613271

北京周立功

地址：北京市海淀区知春路 113 号银网中心 A 座 1207-1208 室（中发电子市场斜对面）
电话：(010)62536178　62536179　82628073
传真：(010)82614433

重庆周立功

地址：重庆市石桥铺科园一路二号大西洋国际大厦（赛格电子市场）1611 室
电话：(023)68796438　68796439
传真：(023)68796439

杭州周立功

地址：杭州市天目山路 217 号江南电子大厦 502 室
电话：(0571)28139611　28139612　28139613
　　　28139615　28139616　28139618
传真：(0571)28139621

成都周立功

地址：成都市一环路南二段 1 号数码同人港 401 室（磨子桥立交西北角）
电话：(028)85439836　85437446
传真：(028)85437896

深圳周立功

地址：深圳市深南中路 2070 号电子科技大厦 C 座 4 楼 D 室
电话：(0755)83781788（5 线）
传真：(0755)83793285

武汉周立功

地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室（华中电脑数码市场）
电话：(027)87168497　87168297　87168397
传真：(027)87163755

上海周立功

地址：上海市北京东路 668 号科技京城东座 7E 室
电话：(021)53083452　53083453　53083496
传真：(021)53083491

西安办事处

地址：西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881296　83063000　87881295
传真：(029)87880865

销售与服务网络（二）

广州致远电子有限公司



地址：广州市天河区车陂路黄洲工业区 3 栋 2 楼

邮编：510660

传真：(020)38601859

网址：www.embedtools.com （嵌入式系统事业部）

www.embedcontrol.com （工控网络事业部）

www.ecardsys.com （楼宇自动化事业部）

技术支持：

CAN-bus：

电话：(020)22644381 22644382 22644253

邮箱：can.support@embedcontrol.com

iCAN 及数据采集：

电话：(020)28872344 22644373

邮箱：ican@embedcontrol.com

MiniARM：

电话：(020)28872684 28267813

邮箱：miniarm.support@embedtools.com

以太网：

电话：(020)22644380 22644385

邮箱：ethernet.support@embedcontrol.com

无线通讯：

电话：(020) 22644386

邮箱：wireless@embedcontrol.com

串行通讯：

电话：(020)28267800 22644385

邮箱：serial@embedcontrol.com

编程器：

电话：(020)22644371

邮箱：programmer@embedtools.com

分析仪器：

电话：(020)22644375 28872624 28872345

邮箱：tools@embedtools.com

ARM 嵌入式系统：

电话：(020)28872347 28872377 22644383 22644384

邮箱：arm.support@zlgmcu.com

楼宇自动化：

电话：(020)22644376 22644389 28267806

邮箱：mjs.support@ecardsys.com

mifare.support@zlgmcu.com

销售：

电话：(020)22644249 22644399 22644372 22644261 28872524

28872342 28872349 28872569 28872573 38601786

维修：

电话：(020)22644245

目 录

1. CLA简介......1
1.1 目的......1
1.2 CLA概述......1
1.3 CLA结构......2
1.4 CLA接口......4
1.4.1 CLA存储器......4
1.4.2 CLA存储器总线......5
1.4.3 共用外设和EALLOW保护......5
1.4.4 CLA任务和中断向量......6
2. CLA配置和调试......7
2.1 CLA初始化......7
2.2 CLA代码调试......8
2.3 CLA非法操作码的行为......9
2.4 CLA复位......9
3. CLA调试......10
3.1 CCS版本......10
3.2 CCS设置......10
3.3 CCS调试......13
4. 免责声明......20

1. CLA 简介

1.1 目的

Piccolo DSC 包括两个系列：

- TMS320F2802x，包括 F280200/20/21/22/23/26/27，最高主频达到 60MHz；
- TMS320F2803x，包括 F28030/31/32/33/34/35，最高主频达到 60MHz，同时 F28033/35 内部集成了一个 32 位浮点控制律加速器（Control Law Accelerator），简称 CLA。

本文致力于详细介绍 CCS v4.x 开发环境下 CLA 的调试方法。

1.2 CLA 概述

控制律加速器（CLA）是一个独立、完全可编程的 32 位浮点数学处理器，它将并行控制环执行功能引入到 C28x 系列器件。CLA 的低中断延迟使得它能即时读取 ADC 采样。这就极大降低了 ADC 采样到输出的延时，实现了更快的系统响应和更高频率的控制回路。通过利用 CLA 来服务对时间要求严格（time-critical）控制回路，主 CPU 就能自由地处理其它诸如通信、诊断之类的系统任务。

CLA 具有如下显著特点：

- 独立的、可编程的 32 位浮点协处理器；
- 运行频率与主 CPU C28x 一致，并具有独立的 8 级流水线；
- 完整的总线结构：
  - 程序地址总线和程序数据总线；
  - 数据地址总线、数据读总线和数据写总线。
- 12 位程序计数器（MPC）；
- 4 个 32 位的结果寄存器（MR0 – MR3）；
- 2 个 16 位的辅助寄存器（MAR0，MAR1）；
- 支持断点调试；
- 支持 IEEE 单精度浮点运算；
  - 单周期浮点加、减、乘法；
  - 单周期浮点比较、取最大值、取最小值；
  - 单周期 1/x、1/sqrt(x)估算；
  - 数据类型转换；
  - 条件分支和调用；
  - 数据装载/存储操作。
- CLA 程序代码可以包含多达 8 个任务或中断服务程序：
  - 每个任务的起始地址通过 MVECT 寄存器来设定；
  - 任务的大小没有限制，只要求任务的大小在 CLA 程序存储空间的范围之内，；
  - 每个任务一次性服务完，中间不会嵌套其它任务；
  - 当任务完成时，任务特定的中断会在 PIE 中标识出来；
  - 当一个任务完成时，自动启动下个优先级最高的挂起任务。

- 任务触发机制：
  - C28x CPU 通过 IACK 指令来触发；
  - Task1~Task7：对应 ADC 或 ePWM 模块中断。例如：
  - Task1：ADCINT1 或 EPWM1_INT；
  - Task2：ADCINT2 或 EPWM2_INT；
  - Task7：ADCINT7 或 EPWM7_INT；
  - Task8：ADCINT8 或 CPU Timer0 来触发的任务。
- 存储器和共用外设：
  - 2 个专用的信息 RAM（Message RAM），供 CLA 和主 CPU 通信使用；
  - C28x CPU 可以将 CLA 程序和数据存储器映射到主 CPU 空间或 CLA 空间；
  - CLA 可以直接访问 ePWM+HRPWM、比较器和 ADC 结果寄存器。

在含有CLA的处理器中，CLA可将CPU解放出来，自动控制外设的运作，达到更高的控制精度以及更好的实时性。有CLA与没有CLA的区别如图 1.1所示。

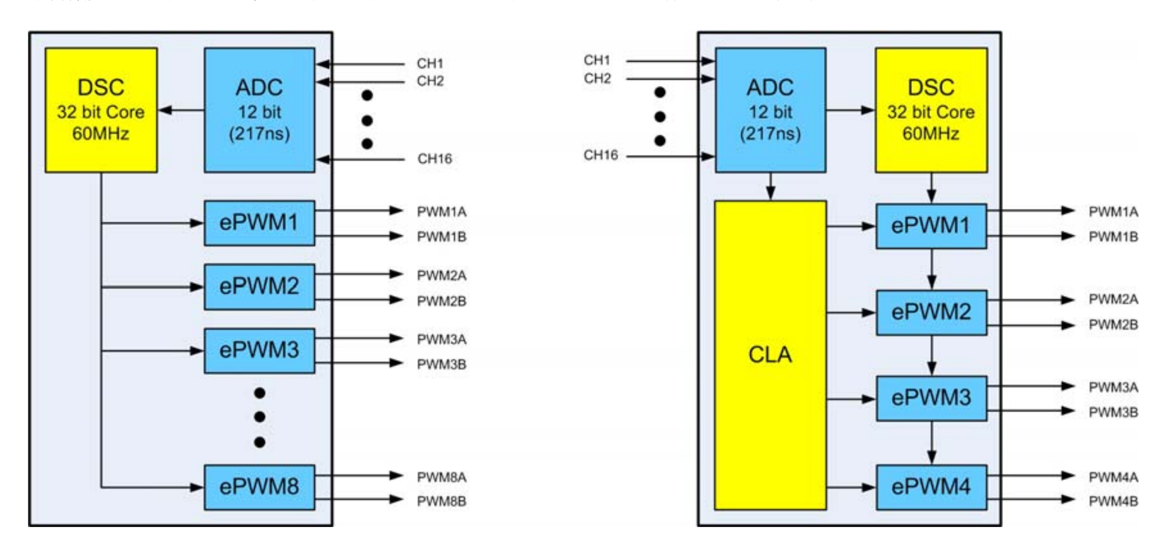


图 1.1　有 CLA 与没有 CLA 的区别

1.3　CLA 结构

CLA 结构有如下特性：

- 减小控制器的响应时间；
- 提高数据传输的响应速度；
- 提供先进的“时序对齐”管理；
- 为系统 IP 释放更多的处理器 MIPS；
- 以相对更低的频率和更低的功耗使处理器执行更多指令；
- 提高采样准确度（无抖动）；
- 使用片内资源提高效率。

如图 1.2所示，为CLA的结构框图。

图 1.2　CLA 结构图

如图 1.3和图 1.4所示分别是无CLA和有CLA时的时序图，可以看出，有CLA时精度明显要比无CLA时要高，同时，在具有CLA的情况下，CLA将CPU解放出来，以便进行其它的处理。

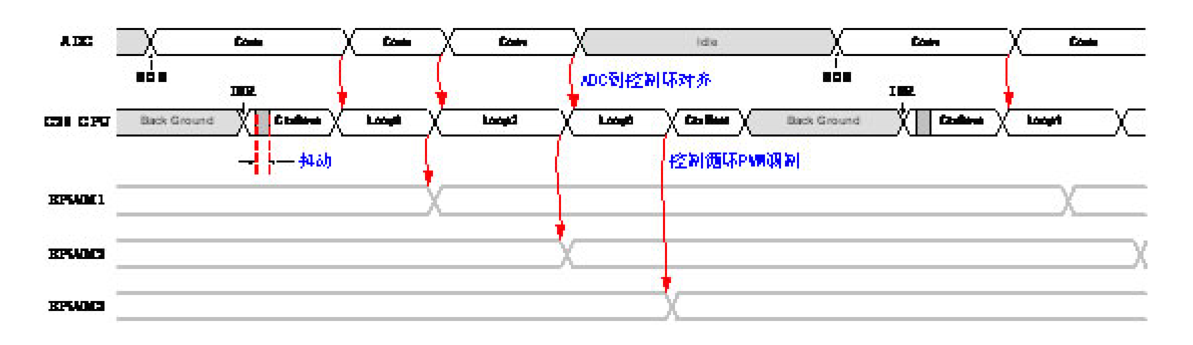


图 1.3　无 CLA 的时序图

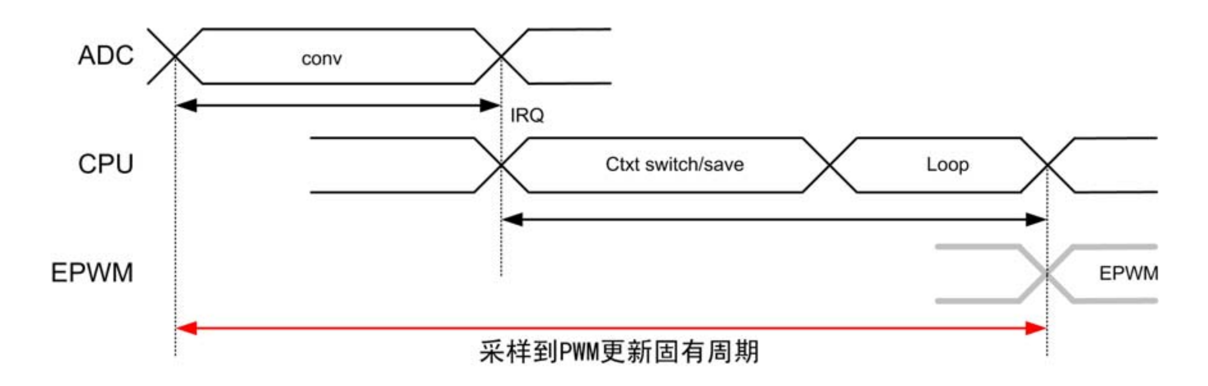


图 1.4　有 CLA 的时序图

对开发者而言，CLA 还有如下特性：

- 可以执行补偿方程（例如 2P2Z IIR filter）；
- 不用 CPU 干预其进程，使用前对其初始化即可；
- 控制位和参数寄存器采用内存映射机制；
- CLA 可由多种方式触发（例如 ADC、PWM 和定时器等）；
- ADC 和 PWM 提供高精度和准确度的时序对齐；
- 不像 CPU，CLA 被触发时不会引起循环开销；
- CLA 没有被触发时，保持低功耗状态；
- CLA 既可以独立操作也可以受控于 CPU。

1.4　CLA 接口

1.4.1 CLA 存储器

CLA 可以访问三类存储器：程序、数据和信息 RAM。每类存储器的操作和仲裁详见附录 A 的描述。

- CLA 程序存储器

复位时，指定用来存放 CLA 程序的存储器映射到主 CPU 存储器中，被当作普通的存储块来对待。当 CLA 程序存储器映射到 CPU 空间时，主 CPU 可以将 CLA 程序代码复制到存储块中。在调试过程中，存储块也可以直接利用 Code Composer Studio 来加载。

一旦存储器被 CLA 代码初始化，主 CPU 就通过写 1 到 MMEMCFG[PROGE]位将其映射到 CLA 程序空间。当存储块被映射到 CLA 程序空间时，CLA 只能对该存储器执行操作码提取操作。当 CLA 停止或空闲时，主 CPU 只能执行调试器访问。如果 CLA 正在执行代码，所有的调试器访问都被终止，存储器读回全零。

CLA 程序存储器受到代码安全模块的保护。所有的 CLA 程序提取操作就相当于执行一系列的 32 位读操作，所有操作码都必须和一个偶地址对齐。由于所有的 CLA 操作码都是 32 位，因此对齐会自然产生。

- CLA 数据存储器

器件上有 2 个 CLA 数据存储块。复位时，两个存储块都被映射到主 CPU 存储空间，CPU 把它们当作普通的存储块来对待。当存储块被映射到 CPU 空间时，主 CPU 可以用数据表和系数将存储块初始化，以备 CLA 使用。

一旦存储器被 CLA 数据初始化，主 CPU 就将其映射到 CLA 空间。两个存储块可以分

别通过 MMEMCFG[RAM0E]和 MMEMCFG[RAM1E]位被映射。当存储块映射到 CLA 数据空间时，CLA 只能对它执行数据访问操作。在这种模式下，主 CPU 只能执行调试器访问。

两个 CLA 数据 RAM 都受到代码安全模块和仿真代码安全逻辑的保护。

■ CLA 共用信息 RAM

有 2 个小的存储块，供 CLA 和主 CPU 两者共享数据和通信使用。两个信息 RAM 总是被映射到 CPU 和 CLA 存储器空间，受代码安全模块保护。只允许对信息 RAM 执行数据访问；不能执行程序提取。

➢ CLA 到 CPU 的信息 RAM

CLA 可以使用这个存储块将数据传递给主 CPU。CLA 可以读和写这个信息 RAM。主 CPU 也可以读该信息 RAM，但写该信息 RAM 的操作会被忽略。

➢ CPU 到 CLA 的信息 RAM

主 CPU 可以使用这个存储块将数据和信息传递给 CLA。主 CPU 可以读和写这个信息 RAM。CLA 可以读该信息 RAM，但写该信息 RAM 的操作会被忽略。

1.4.2 CLA 存储器总线

CLA 有专用的总线结构，与 C28x CPU 的总线结构类似，包含一个程序读、数据读和数据写总线。这样，就可以在一个周期内同时执行指令提取、数据读和数据写操作。和 C28x CPU 类似，CLA 希望存储器逻辑使所有 32 位的读和写都与一个偶地址对齐。如果地址-产生逻辑产生一个奇地址，CLA 将在前一个偶地址处开始读或写。这种对齐不影响地址-产生逻辑产生的地址值。

■ CLA 程序总线

CLA 程序总线可以访问多达 2048 条 32 位的指令。由于所有的 CLA 指令都是 32 位的，所以，这个总线总是一次提取 32 个位，而且，操作码必须是偶数字对齐的。CLA 可以使用的存储空间由具体的器件决定，请参考器件特定的数据手册。

■ CLA 数据读总线

CLA 数据读总线的地址范围为 64K×16。总线可以执行 16 或 32 位读，并且，如果出现存储器访问冲突，总线将自动停止。数据读总线可以访问信息 RAM、CLA 数据存储器以及 ePWM、HRPWM、比较器和 ADC 结果寄存器。

■ CLA 数据写总线

CLA 数据写总线的地址范围为 64K×16。总线可以执行 16 或 32 位写。如果出现存储器访问冲突，总线将自动停止。数据写总线可以访问 CLA 到 CPU 的信息 RAM、CLA 数据存储器以及 ePWM、HRPWM 和比较器结果寄存器。

1.4.3 共用外设和 EALLOW 保护

CLA 和主 CPU 都能访问 ePWM、HRPWM、比较器和 ADC 结果寄存器。当 CLA 和 CPU 都要访问这些结果寄存器时，需要对 CLA 和 CPU 进行仲裁，详细的描述请见**错误！未找到引用源。**。

EALLOW 保护机制可以保护几个外设控制寄存器，防止 28x CPU 对其执行虚假的写操作。这些寄存器还受到保护，以免发生 CLA 虚写。主 CPU 状态寄存器 1（ST1）的 EALLOW 位指示了主 CPU 的保护状态。同样地，CLA 状态寄存器（MSTF）的 MEALLOW 位指示了 CLA 写保护的状态。MEALLOW CLA 指令使能 CLA 写受 EALLOW 保护的寄存器。类似地，MEDIS CLA 指令禁止 CLA 写这些寄存器。这样，就能单独使能/禁能 CLA 的写访问，与主

CPU 无关。

F2803x ADC 可以选择在 ADC 开始转换时产生一个预先中断脉冲。如果利用它来启动一个 ADC 触发的 CLA 任务，那么转换一结束第 8 条指令就能读出转换结果。

1.4.4 CLA 任务和中断向量

CLA 程序代码被分割成任务或中断服务程序。任务没有固定的起始位置或长度。CLA 程序存储器可以根据需要被分割。CLA 根据相应的中断向量（MVECT1~MVECT8）的内容来判定任务从何处开始，任务的结束通过 MSTOP 指令来指示。

CLA 支持 8 个任务。Task1 的优先级最高，Task8 的优先级最低。一个任务可以由外设中断或软件来请求：

- 外设中断触发的任务

每个任务都有特定的中断源来触发它。配置 PERINT1SEL 寄存器从可能的中断源中选择一个来触发任务。例如，Task 1（MVECT1）可以由 ADCINT1 或 EPWM1_INT（在 PERINT1SEL 中指定）来触发。但是，你不能使用 EPWM2_INT 来直接触发 Task 1。如果你要使用 EPWM2_INT 来触发一个任务，最好的方法是使用 Task 2（MVECT2）。另一个可行的方法是让主 CPU 使用 EPWM2_INT，由软件来触发任务。

如果要禁止外设向 CLA 发送中断请求，就将 PERINT1SEL 设置成无中断源。

- 软件触发的任务

任务也可以通过主 CPU 软件写 MIFRC 寄存器或通过 IACK 指令来触发。使用 IACK 指令更高效，因为不需要发布一个 EALLOW 来设置 MIFR 的位。IACK 特性通过设置 MCTL[IACKE]位来使能。IACK 指令操作数的每个位对应一个任务。例如，IACK #0x0001 将设置 MIFR 寄存器的位 0 来启动 Task 1。类似地，IACK #0x0003 将设置 MIFR 寄存器的位 0 和位 1 来启动 Task 1 和 Task 2。

CLA 有自己的提取机制，可以独立运行和执行一个任务，与主 CPU 无关。一次只能服务一个任务，没有任务嵌套。当前正在运行的任务在 MIRUN 寄存器中指示出来。已经接收到但还没有被服务的中断在标志寄存器（MIFR）中标识出来。如果接收到一个外设的中断请求，而这个任务又已经被标识出来了，那么就将溢出标志位置位。溢出标志将保持置位，直到被主 CPU 清除。

如果 CLA 空闲（当前没有任务正在运行），已经被标识出来（MIFR）且已使能（MIER）的最高优先级中断将启动。整个流程如下：

- 相应 RUN 寄存器中的位被置位（MIRUN），标志位（MIFR）被清除；
- CLA 开始从相应中断向量（MVECTx）指定的位置开始执行。MVECT 是基于起始程序存储器单元的一个偏移量；
- CLA 执行指令，直至找到 MSTOP 指令，表明任务结束；
- 清除 MIRUN 位；
- 向 PIE 提交任务特定的中断。告知主 CPU 任务已经结束；
- CLA 返回到空闲状态。

一旦一个任务结束，下个优先级最高、挂起的任务将自动被服务。然后再重复上面这个过程。

2. CLA 配置和调试

CLA 有独立的指令系列，必须使用 CLA 汇编语言对控制律加速器进行编程，即 CLA 不支持 C 语言。CLA 汇编代码可以、也应该和 C28x 代码位于同一个工程中。唯一的限制是 CLA 代码必须位于自己的汇编段（Assembly Section）内。这可以利用.sect 汇编命令轻松做到。而且，这样做不妨碍 CLA 和 C28x 代码被连接到链接器命令文件的相同存储区内。

系统和 CLA 的初始化由主 CPU 来执行。初始化可能通常用 C 或 C++代码来实现，但也可以包含 C28x 汇编代码。主 CPU 还将把 CLA 代码复制到程序存储器中，而且，如果需要，还将初始化 CLA 数据 RAM。一旦系统初始化完成，程序开始执行。CLA 将使用自己的汇编代码来服务自己的中断（或任务）。与此同时，主 CPU 可以处理其他任务。

如果设置了开关：-- cla_support = cla0，V5.2.x 以及更高版本的 C2000 代码产生工具将支持 CLA 指令。

CLA 指令系统可以参考相关数据手册。

2.1 CLA 初始化

典型的 CLA 初始化序列由主 CPU 来执行，具体请见本节的描述。

1. 将 CLA 代码复制到 CLA 程序 RAM 中

CLA 代码源可以一开始位于 Flash 中，或者位于一个通信外设的数据流内，或者在主 CPU 能访问它的任何地方。在开发过程中，也可以利用调试器将代码直接加载到 CLA 程序 RAM 中。

2. 初始化 CLA 数据 RAM（如果需要）

将任何需要的数据系数或常数填充到 CLA 数据 RAM 中。

3. 配置 CLA 寄存器

配置 CLA 寄存器，中断必须禁能（MIER == 0），直到之后执行以下操作。

■ 在 PCLKCR3 寄存器中使能 CLA 时钟

定义 PCLKCR3 寄存器，请参考器件特定的系统控制和中断参考指南。

■ 填充 CLA 任务中断向量：MVECT1 ~ MVECT8

当 CLA 接收到相应的中断时，每个向量初始化成要执行任务的起始地址。这个地址是一个基于 CLA 程序存储器首个地址的偏移量。例如，0x0000 对应首个 CLA 程序存储器地址。

■ 选择任务中断源

在 PERINT1SEL 寄存器中为每个任务选择中断源。如果任务将由软件来产生，就选择没有中断。

■ 使能 IACK 启动一个软件产生的任务（如果希望）

为了使能 IACK 指令来启动一个任务，置位 MCTL[IACKE]位。使用 IACK 指令就无需再执行置位和清除 EALLOW 位的操作。

■ 映射 CLA 数据 RAM 到 CLA 空间（如果必要）

通过写 1 到 MMEMCFG[RAM0E]和 MMEMCFG[RAM1E]位将其中一个或全部两个数据 RAM 映射到 CLA 空间。在存储器映射到 CLA 空间之后，主 CPU 就不能再访问它了。允许在改变这个存储器的映射配置和访问该存储器之间存在 2 个 SYSCLKOUT 周期的时间。

■ 映射 CLA 程序 RAM 到 CLA 空间

通过置位 MMEMCFG[PROGE]位将 CLA 程序 RAM 映射到 CLA 空间。在存储器重新映射到 CLA 空间之后，主 CPU 只能对存储块进行调试访问。允许在改变这些存储器的映射配置和访问它们之间存在 2 个 SYSCLKOUT 周期的时间。

4. 初始化 PIE 向量表和寄存器

当一个 CLA 任务结束时，PIE 中相应的中断将被标识出来。CLA 上溢和下溢标志也在 PIE 中对应有相应的中断。

5. 使能 CLA 任务/中断

置位中断使能寄存器（MIER）中相应的位来使能 CLA 服务中断。

6. 初始化其它外设

将那些要向 CLA 产生中断以及将被一个 CLA 任务服务的所有外设（ePWM、ADC 等等）初始化。

CLA 现在已经做好服务中断的准备，而且信息 RAM 可以被用来执行 CPU 和 CLA 之间的数据传递。通常 CLA 程序和数据 RAM 的映射只在初始化过程中出现。如果在一段时间之后你想将这些存储器重新映射回 CPU 空间，必须先禁能中断，并通过检查 MIRUN 寄存器确保所有的任务已经结束。通常允许在改变这些存储器的映射配置和访问它们之间存在 2 个 SYSCLKOUT 周期的时间。

2.2 CLA 代码调试

调试 CLA 代码是一个简单的过程，独立出现，与主 CPU 无关。

1. 在 CLA 代码中插入一个断点

在希望 CLA 停止的地方插入一个 CLA 断点（MDEBUGSTOP 指令）。然后重新构建和重新加载代码。由于在单步调试时 CLA 不清理流水线，因此必须将 MDEBUGSTOP 指令作为代码的一部分插入。调试器不能根据需要插入该指令。

如果不允许单步执行，MDEBUGSTOP 将被忽略，当做一个 MNOP 来处理。MDEBUGSTOP 指令可以放置在 CLA 代码的任何地方，只要不在 MBCNDD、MCCNDD 或 MRCNDD 的前 3 条和后 3 条指令以内。

2. 使能 CLA 单步执行

首先使能调试器中的 CLA 单步执行。单步执行在复位时被禁止。

3. 启动任务

当任务启动时，CLA 执行指令，直至 MDEBUGSTOP 处在流水线的 D2 级。这时，CLA 将停止，流水线暂停。MPC 寄存器将反映出 MDEBUGSTOP 指令的地址。

4. 单步执行 CLA 代码

一旦 CLA 停止，你就能单步执行 CLA 代码，一次执行一个周期。CLA 的单步执行操作与主 C28x 的不同。当发起一次 CLA 单步执行时，只给流水线提供一个周期的时钟，然后再次暂停。而在 28x CPU 上，每执行一次单步调试都要清理流水线。

注：当 CLA 程序存储器映射到 CLA 存储器空间时，CLA 提取操作的优先级高于 CPU 调试读。由于这个原因，如果 CLA 正在一个循环中执行，CLA 就可能永久地阻止 CPU 调试访问。这种情况很可能出现在 CLA 代码开发的初始阶段，在这个阶段代码很容易由于 bug（程序缺陷）而进入一个死循环。为了避免锁定主 CPU 的情况发生，当 CLA 正在运行时，执行 CPU 调试读程序存储器将返回全零。当 CLA 停止或空闲时，可以对 CLA 程序存储器执行正常的 CPU 调试读和写访问。

如果 CLA 进入一个死循环，可以使用一个软和硬复位来退出这种状态。也可以利用调试器复位来退出这种状态。

当单步调试一个任务使得程序计数器 MPC 到达任务末尾处的 MSTOP 指令时，可能会出现一些特殊情况。

- MPC 在 MSTOP 指令处或在 MSTOP 指令之后停止，此时已经有一个任务挂起

如果你正在单步调试“Task A”或者停止了“Task A”处理的时候“Task B”到来，但在“Task A”中 MPC 并未到达 MSTOP 指令，那么，如果你要继续单步执行 MSTOP 指令，“Task B”就会启动。基本上，如果在“Task A”的 MPC 到达 MSTOP 之前“Task B”正在挂起，“Task B”就必定启动，不需要任何特殊操作。

- MPC 在 MSTOP 指令处或在 MSTOP 指令之后停止，此时没有任务正在挂起

在这种情况下，你已经单步调试了“Task A”或停止了“Task A”的处理，而且，MPC 已经到达了 MSTOP 指令，但没有任务正在挂起。如果这时“Task B”到来，它将在 MIFR 寄存器中标识出来，但是，如果你继续单步执行“Task A”的 MSTOP 指令，“Task B”可能启动，也可能不启动。

这完全取决于新任务何时到来。为了可靠地启动“Task B”，可以执行一次软复位以及重新配置 MIER 位。一旦这样做，你就可以开始单步调试“Task B”。

如果“Task B”何时到来受到控制（例如，使用 IACK 指令来启动任务），处理起来就稍微有点不同。这时，你已经单步调试了“Task A”或停止了“Task A”的处理，而且，MPC 已经到达了 MSTOP 指令，但没有任务正在挂起。在强制“Task B”启动之前，空运转来迫使 CLA 退出调试状态。一旦这样做，你就可以强制启动“Task B”并继续调试。

2.3 CLA 非法操作码的行为

如果 CLA 提取一个对应一条非法指令的操作码，操作如下：

- 随着非法操作码在流水线的 D2 级出现，CLA 将停止，就好像有一个断点似的。不管 CLA 单步调试使能与否，这种情况都会出现；
- CLA 将向 PIE 提交任务特定的中断；
- 任务的 MIRUN 位将保持置位。

一旦程序的执行因为一个非法操作码而终止，后面的单步调试就被忽略。要退出这种状态，可以启动一个 CLA 软或硬复位。

2.4 CLA 复位

如果需要复位 CLA 时，可能要执行多次复位。例如，在代码调试过程中，CLA 可能由于一个代码 bug 而进入一个死循环。CLA 有 2 种类型的复位：硬复位和软复位。两种复位都可以由调试器或主 CPU 来执行。

- 硬复位

写 1 到 MCTL[HARDRESET]位来执行一次 CLA 硬复位。硬复位的操作与系统复位（通过 $\overline{XRS}$ 或调试器来实现）相同。这种情况下，所有 CLA 配置和执行寄存器都被设置成它们的默认状态，CLA 执行终止。

- 软复位

写 1 到 MCTL[SOFTRESET]位来执行一次 CLA 软复位。如果正在执行一个任务，任务将停止，且相应的 MIRUN 位被清除。也可以清除中断使能寄存器（MIER）中的所有位，导致不启动新任务。

3. CLA 调试

3.1 CCS 版本

目前 TI 官方网站上可下载到 CCS v4.0.2，包括白金版和微处理器版：

- 白金版：适用于 DSK、XDS560 和 XDS100。适合处理器：TMS320C6000、TMS320C5000、TMS320C2800、TMS470、TMS570、ARM7、ARM9、ARM11、ARM Cortex M3、ARM Cortex R4、ARM Cortex A8 和 MSP430；
- 微处理器版：适用于 XDS100。适合处理器：32KB IDE（适用于 C28x），16KB IDE（适用于 MSP430）。

需要使用CCS v4.x的用户可以自行到TI官方网站下载：www.ti.com.cn。

CCS v4.x的快速入门文档可以到周立功官方网站下载：www.zlgmcu.com。

这里介绍微处理器版 CCS v4.x 开发环境下 CLA 的调试。

运行CCS v4.0.2，选择【Help | Software Updatas | Manage Configuration】，查看软件版本：4.0.2，Flash版本：4.0.3，如图 3.1所示。

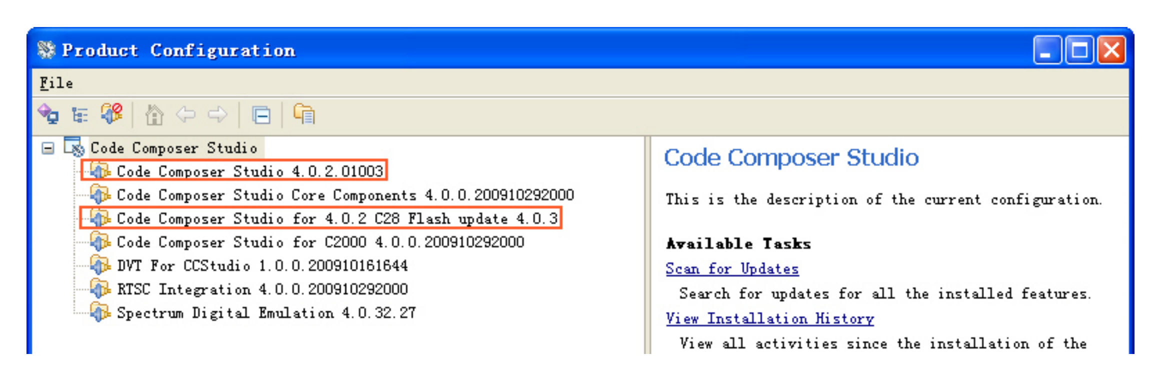


图 3.1　CCS v4.x 版本

EasyDSP28022/EasyDSP28035 光盘里没有放置 CCS v4.x 的软件，所以需要用户自己下载。同时购买了 EasyDSP28022/EasyDSP28035 的用户需要多注意 TI 网站上是否有最新的 F2802x/F2803x 的文档更新。

3.2 CCS 设置

先到 TI 网站下载最新的 F2803x 头文件及例程的安装包【setup_DSP2803x_v121.exe】，版本号为 v1.21，并且按默认路径安装。

运行 CCS，在 CCS 窗口菜单栏选择【Project | Import existing CCS/CCE Eclipse Project】，打开【C:\tidcs\c28\DSP2803x\v121\DSP2803x_examples_ccsv4】路径下的【cla_adc_fir】的工程。

由于【cla_adc_fir】工程文件夹下没有相应的目标配置文件，可以参考《CCS v4.x快速入门》文档介绍，新建目标配置文件，这里就不在赘述了，如图 3.2所示。

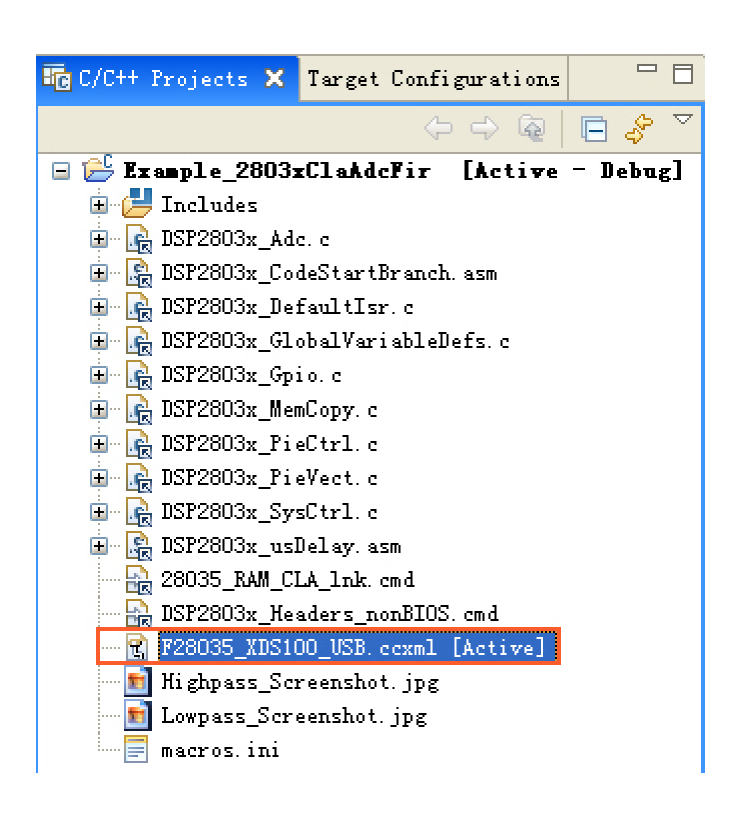


图 3.2　打开工程

右键工程名，在菜单中选择【Build Properties】，选择【C2000 Compiler】下的【Runtime Model Options】，在【Specify CLA support】选择【cla0(default)】，如图 3.3所示。要调试CLA，必须选中cla0。

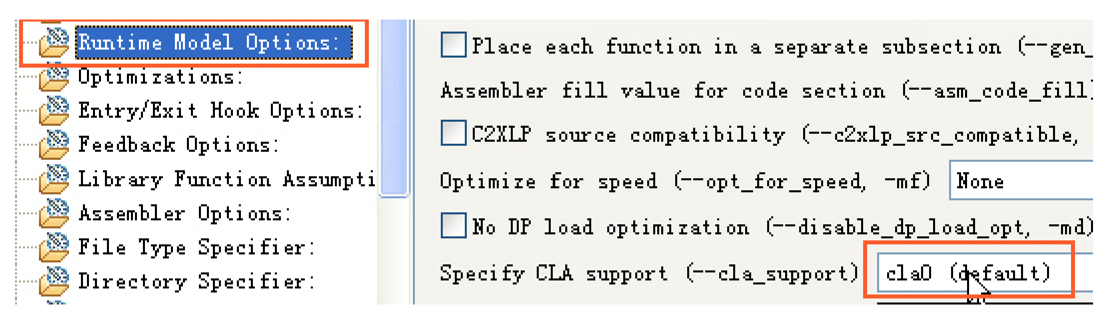


图 3.3　选中 cla0

在CCS左侧栏中打开【28035_RAM_CLA_lnk.cmd】文件，在【MEMORY | PAGE 1】段增加了【CLA1_MSGRAMLOW】和【CLA1_MSGRAMHIGH】，如图 3.4所示。在【SECTIONS】段增加了【Cla1Prog】、【Cla1ToCpuMsgRAM】和【CpuToCla1MsgRAM】，如图 3.5所示。

【CLA1_MSGRAM】是主 CPU 和 CLA 的信息 RAM。【Cla1Prog】是把 CLA 代码拷贝到 CLA 可运行的 RAM 段，具体可见 F2803x 数据手册存储器分配那部分的内容。

在 Debug 模式下程序都放在 RAM，所以在链接时直接将这部分代码拷贝到 CLA 可运行的 RAM 段；在 Release 模式下，所有的代码都放在 Flash 中，所以必须由主 CPU 在初始化的时候将 CLA 代码拷贝到 CLA 可运行的 RAM 段。

Debug 模式下的代码可参考例程【cla_adc_fir】。

```
MEMORY
{
PAGE 0 :
   /* BEGIN is used for the "boot to SARAM" bootloader mode    *

   BEGIN                  : origin = 0x000000, length = 0x000002
   RAMM0                  : origin = 0x000050, length = 0x0003B0
   RAMLOL1                : origin = 0x008000, length = 0x000C00
   RAML3                  : origin = 0x009000, length = 0x001000
   RESET                  : origin = 0x3FFFC0, length = 0x000002
   IQTABLES               : origin = 0x3FE000, length = 0x000B50
   IQTABLES2              : origin = 0x3FEB50, length = 0x00008C
   IQTABLES3              : origin = 0x3FEBDC, length = 0x0000AA

   BOOTROM                : origin = 0x3FF27C, length = 0x000D44


PAGE 1 :

   BOOT_RSVD              : origin = 0x000002, length = 0x00004E
   RAMM1                  : origin = 0x000480, length = 0x000380
   RAML2                  : origin = 0x008C00, length = 0x000400
   CLA1_MSGRAMLOW         : origin = 0x001480, length = 0x000080
   CLA1_MSGRAMHIGH        : origin = 0x001500, length = 0x000080
}
```

图 3.4 cmd 文件 memory 段

```
SECTIONS
{
   /* Setup for "boot to SARAM" mode:
      The codestart section (found in DSP28_CodeStartBranch.asm)
      re-directs execution to the start of user code.   */
   codestart          : > BEGIN,      PAGE = 0
   ramfuncs           : > RAMM0       PAGE = 0
   .text              : > RAMLOL1,    PAGE = 0
   .cinit             : > RAMM0,      PAGE = 0
   .pinit             : > RAMM0,      PAGE = 0
   .switch            : > RAMM0,      PAGE = 0
   .reset             : > RESET,      PAGE = 0, TYPE = DSECT /* not used, */

   .stack             : > RAMM1,      PAGE = 1
   .ebss              : > RAML2,      PAGE = 1
   .econst            : > RAML2,      PAGE = 1
   .esysmem           : > RAML2,      PAGE = 1

   IQmath             : > RAMLOL1,    PAGE = 0
   IQmathTables       : > IQTABLES,   PAGE = 0, TYPE = NOLOAD

   Cla1Prog           : LOAD = RAMLOL1,
                        RUN = RAML3,
                        LOAD_START(_Cla1funcsLoadStart),
                        LOAD_END(_Cla1funcsLoadEnd),
                        RUN_START(_Cla1funcsRunStart),
                        PAGE = 0

   Cla1ToCpuMsgRAM    : > CLA1_MSGRAMLOW,   PAGE = 1
   CpuToCla1MsgRAM    : > CLA1_MSGRAMHIGH,  PAGE = 1
```

图 3.5 cmd 文件 section 段

下面介绍 Release 模式下相关的代码的更改。

在【C:\tidcs\c28\DSP2803x\v121\DSP2803x_common\cmd】路径下F28035.cmd文件中【MEMORY | PAGE 1】段增加【CLA1_MSGRAMLOW】和【CLA1_MSGRAMHIGH】，如图3.4所示。【SECTIONS】段增加【Cla1Prog】、【Cla1ToCpuMsgRAM】和【CpuToCla1MsgRAM】，如图 3.5所示。不过必须将【Cla1Prog】部分改成如下形式：

```
Cla1Prog            : LOAD = FLASHA,
                      RUN = RAML3,
                      LOAD_START(_Cla1funcsLoadStart),
                      LOAD_END(_Cla1funcsLoadEnd),
                      RUN_START(_Cla1funcsRunStart),
                      PAGE = 0
```

3.3 CCS 调试

编译工程，直接单击图标进入调试环境，单击图标加载程序，如图 3.6所示。此时调试光标指向了main函数。

在左上角的【Debug】窗口选择【Texas Instruments XDS100 USB Emulator_0/CLA0】，然后在右键菜单中选择【Connect Target】，如图 3.7所示；或单击菜单栏的图标连接CLA内核，如图 3.8所示。

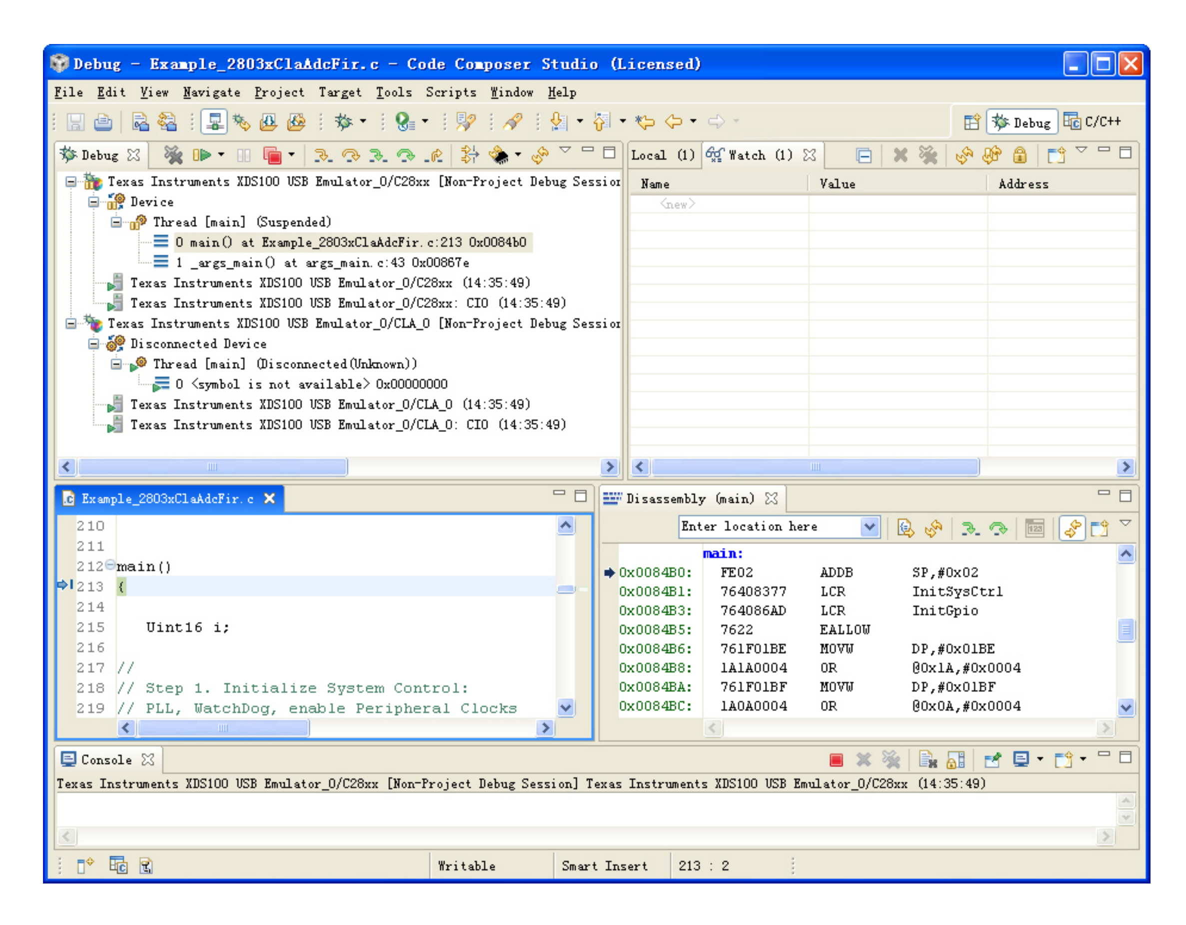


图 3.6 调试窗口

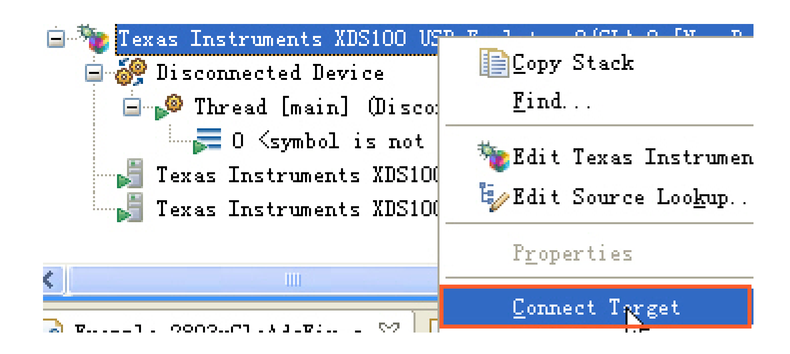


图 3.7　连接目标



图 3.8　连接目标

若不想每一次进入调试环境时总是去连接目标板，可以选择【Target | Debug】，按如图3.9所示进行设置。然后退出调试环境，重新插拔USB电缆，单击图标再次进入调试环境，这样就不必手动连接目标板。

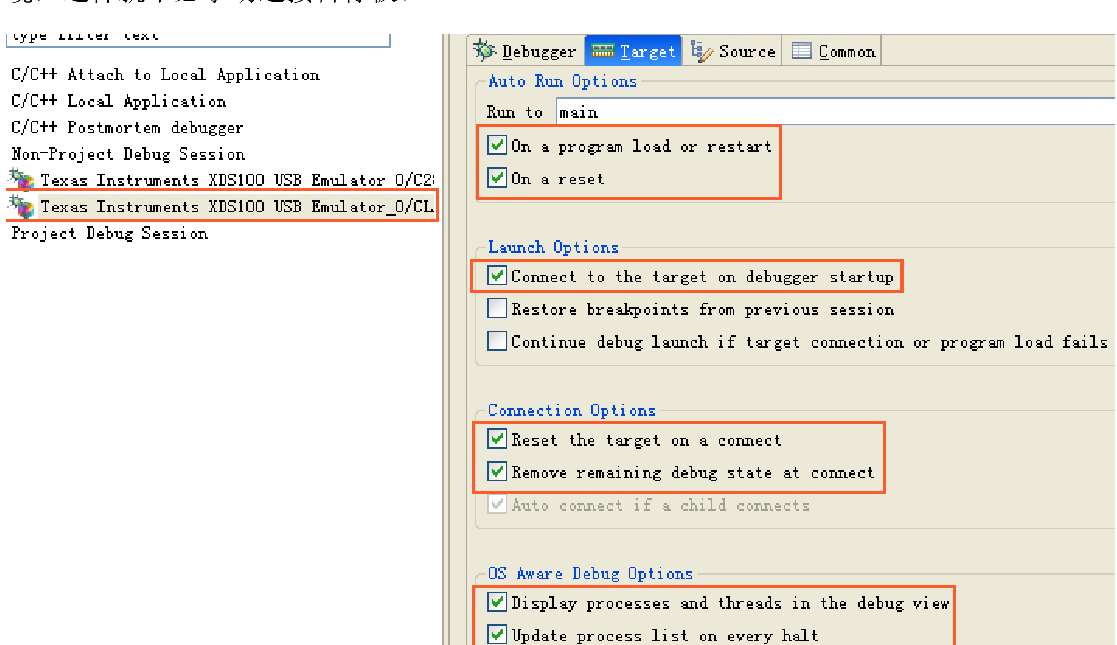


图 3.9　设置 Debug

单击左下角或右上角的图标，打开【CLA_FIR.asm】文件，如所示。

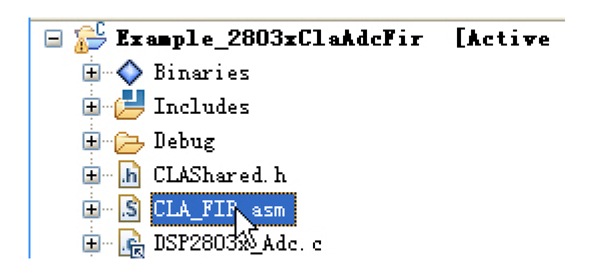


图 3.10　打开文件

选中【Debug】窗口的【0 main0 at Example_2803xClaAdcFir.c:240 0x0084be】，如图 3.11 所示。对应的反汇编窗口将出现C28x内核的汇编代码，如图 3.12所示。

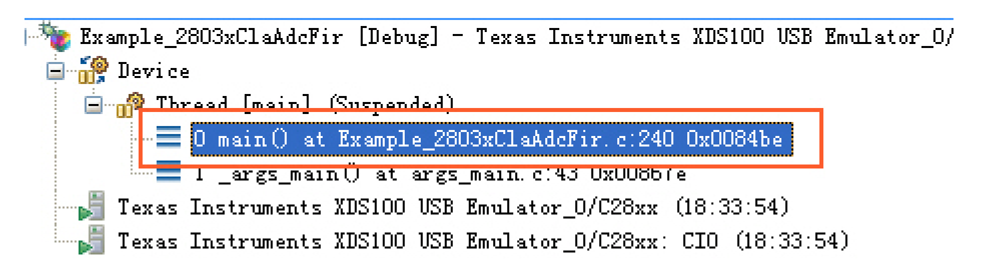


图 3.11　查看 C28x 汇编代码

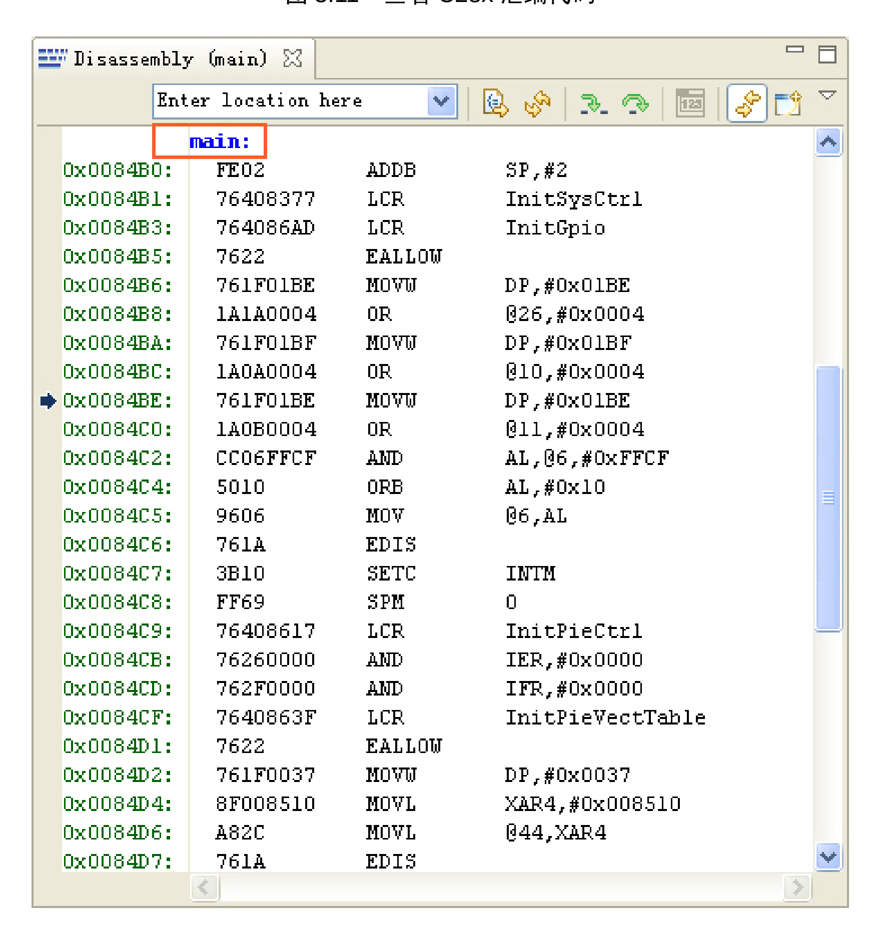


图 3.12　C28x 汇编代码

选中【Debug】窗口的【0 <.symbol is not available> 0x00009000】，如图 3.13所示。对应的反汇编窗口将出现CLA内核的汇编代码，如图 3.14所示。

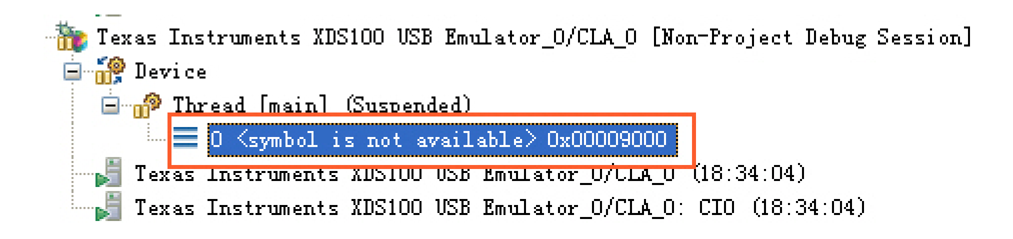


图 3.13　查看 CLA 汇编代码

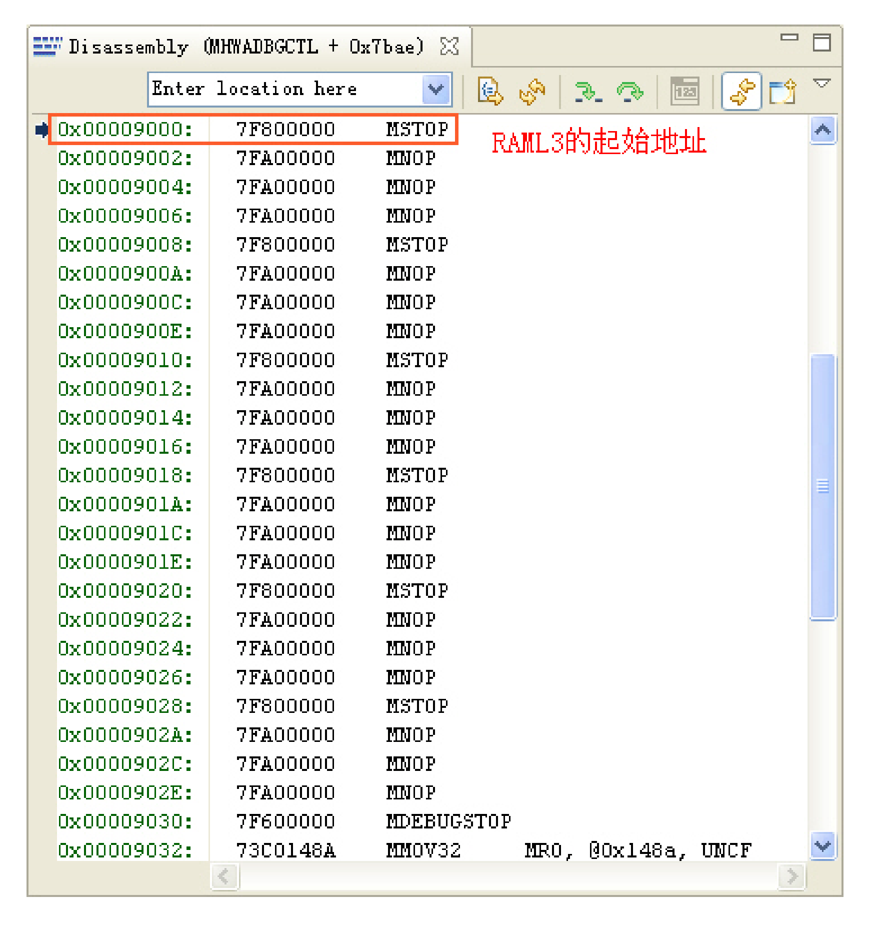


图 3.14 CLA 汇编代码

【0 main0 at Example_2803xClaAdcFir.c:240 0x0084be】和【0 <.symbol is not available> 0x00009000】中的 0x0084be 和 00009000 分别是 C28x 和 CLA 内核代码的当前指令的地址。

在【Example_2803xClaAdcFir.c】源文件的【interrupt void cla1_isr7()】函数放置断点，如图 3.15所示。

```
390
391        AdcRegs.ADCINTFLGCLR.bit.ADCINT7 = 1;
392        PieCtrlRegs.PIEACK.all = 0xFFFF;
393        GpioDataRegs.GPATOGGLE.bit.GPIO18 = 1;
```

图 3.15 放置断点

CLA调试窗口不允许放置断点，只能靠放置指令【MDEBUGSTOP】来代替断点，详细见【CLA代码调试】。

在C28x内核的调试窗口单击 或按【F8】全速运行，此时调试光标会在CLA内核的汇编窗口的【MDEBUGSTOP】指令处停下来，如图 3.16所示。

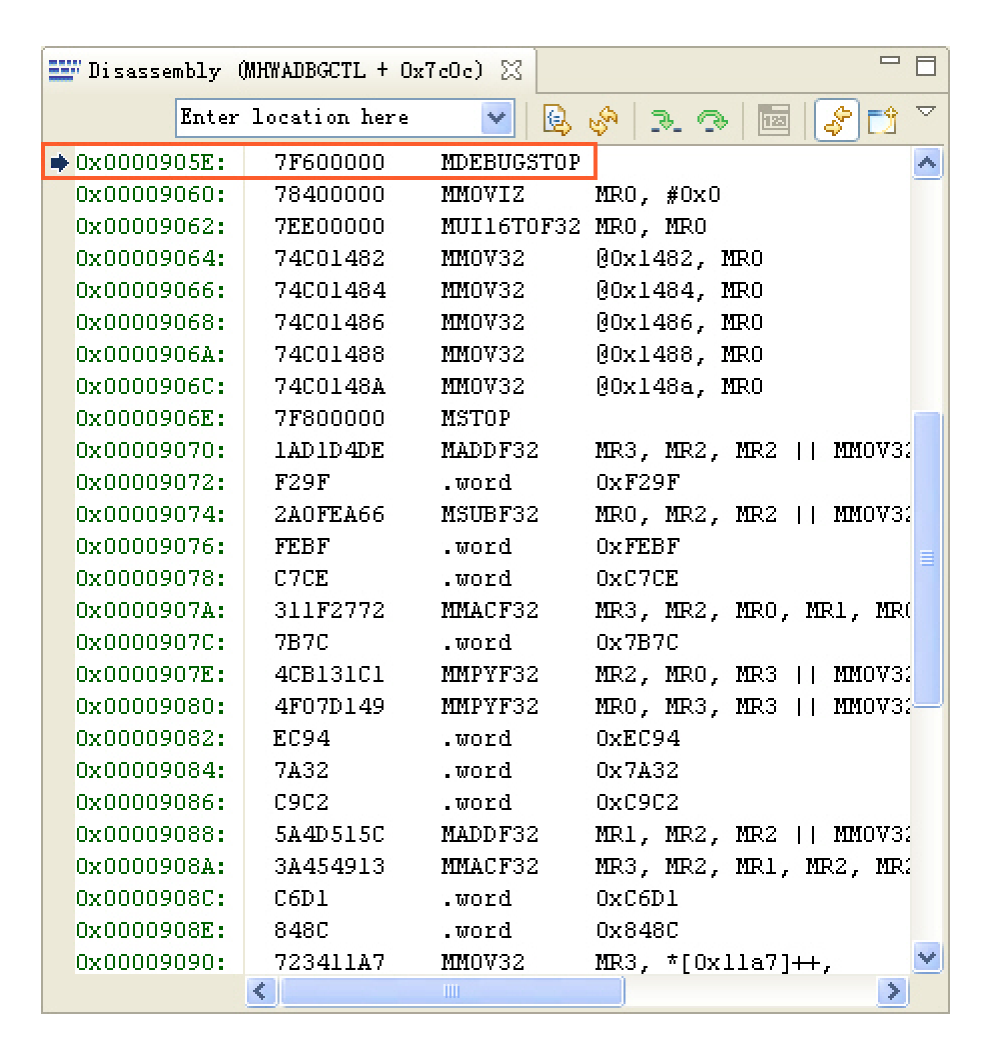


图 3.16　CLA 代码调试

在【Debug】窗口选中【0 <.symbol is not available> 0x00009000】，此时【Debug】窗口的调试图标变为可用，单击 或 ，或者【Disassembly】窗口 图标可进行单步调试，如所示。

CLA支持单步调试，如图 3.17所示。

在CLA内核【Debug】窗口单击 或按【F8】全速运行，CLA全速运行并且在【MSTOP】指令处停止时，C28x内核的调试窗口的调试光标会在我们放置了断点的地方停止，如图 3.18 所示。

C28x 内核就可以完全不用去读 ADC 的结果，只要在函数【interrupt void cla1_isr7()】清除一下标志位就可以了，每次 CLA 完成了 FIR 滤波等操作之后再通知 C28x 内核。这样可以节省很多 C28x 的的时间，大大的提高了处理器的性能。

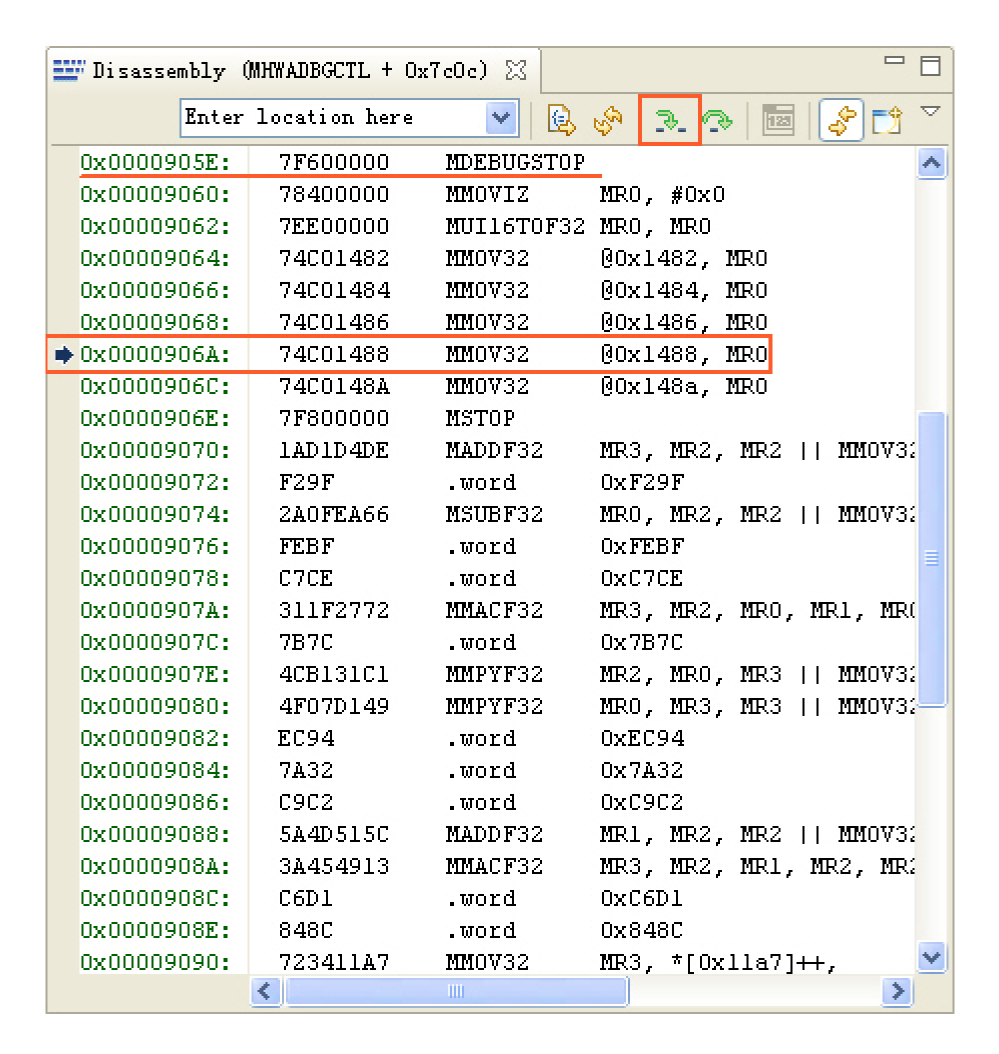


图 3.17　CLA 单步调试

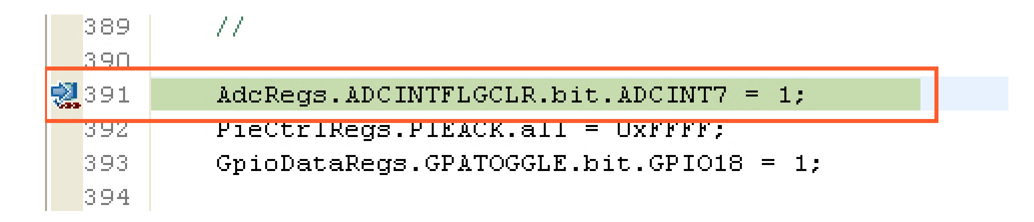


图 3.18　C28x 在断点处停止

如果程序没有运行到CLA的代码或者停在【MSTOP】指令处，却在CLA调试窗口进行调试操作，那么CCS会出现如图 3.19所示的界面。

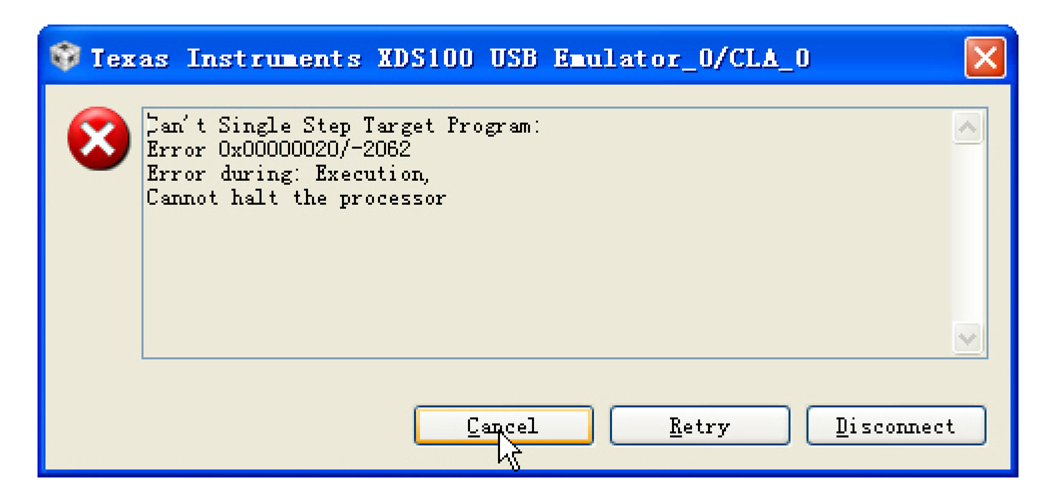


图 3.19　CLA 调试报错

至此，CCS v4.x 开发环境下 C28x 和 CLA 双核调试就算完成了。更多的细节需要用户在调试过程中慢慢体会摸索。关于 CLA 更多的介绍可以参考 CLA 的数据手册。

注意：

- CCS v4.0.1 版不支持 CLA；
- CLA 有独立的汇编语言，不支持 C 语言；
- CLA 支持单步调试，但不支持断点，可以使用【MDEBUGSTOP】指令代替断点；
- CLA停止运行的时候不能进行调试，否则会出现图 3.19的错误提示。

4. 免责声明

广州周立功单片机发展有限公司随附提供的软件或文档资料旨在提供给您（本公司的客户）使用，仅限于且只能在本公司制造或销售的产品上使用。

该软件或文档资料为本公司和/或其供应商所有，并受适用的版权法保护。版权所有。如有违反，将面临相关适用法律的刑事制裁，并承担违背此许可的条款和条件的民事责任。本公司保留在不通知读者的情况下，修改文档或软件相关内容的权利，对于使用中所出现的任何效果，本公司不承担任何责任。该软件或文档资料“按现状”提供。不提供保证，无论是明示的、暗示的还是法定的保证。这些保证包括（但不限于）对出于某一特定目的应用此软件的适销性和适用性默示的保证。在任何情况下，公司不会对任何原因造成的特别的、偶然的或间接的损害负责。